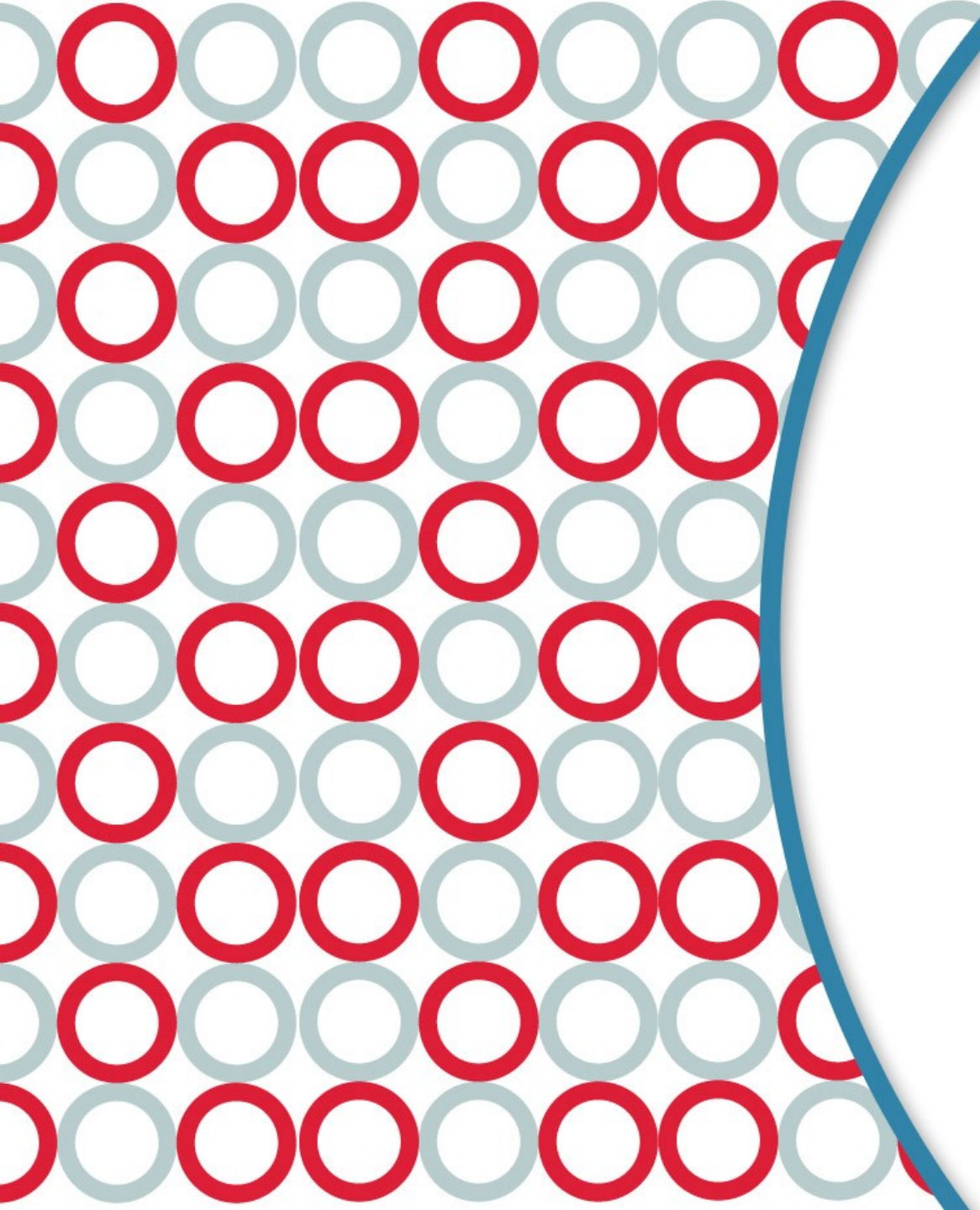


May 2019, IPT Course
Java Web Debelopment

Remote Method Invocation (RMI)

Trayan Iliev
tiliev@iproduct.org
http://iproduct.org

Where to Find the Code?

Intarmediate Java Programming projects and examples are available @ GitHub:

https://github.com/iproduct/course-java-web-development

Agenda for This Session

- RMI architecture
- Dynamic code loading
- Remote Interfaces, objects, and methods
- Using SecurityManager, security permissions and policies for accessing remote code
- Exporting the remote object using UnicastRemoteObject class
- Using RMI registry
- Remote exception handling
- Building sample RMI client-server application – ComputeEngine

Software Architecture

Fundamental organization of a system implemented through its components, the relationships between them and with the environment, as well as the principles guiding their design and evolution.

[ANSI / IEEE]

A set of important decisions about organization of a software system, the choice of structural elements and their interfaces, by which the system is composed, together with their behavior, as specified by collaborations between these structural and behavioural elements in progressively larger subsystems, and architectural styles guiding this organization.

[Booch, Rumbaugh, Jacobson, Unified Process]

Software Architecture

A software architecture is an abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture.

A software architecture is defined by a configuration of architectural elements—components, connectors, and data—constrained in their relationships in order to achieve a desired set of architectural properties.

[Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures]

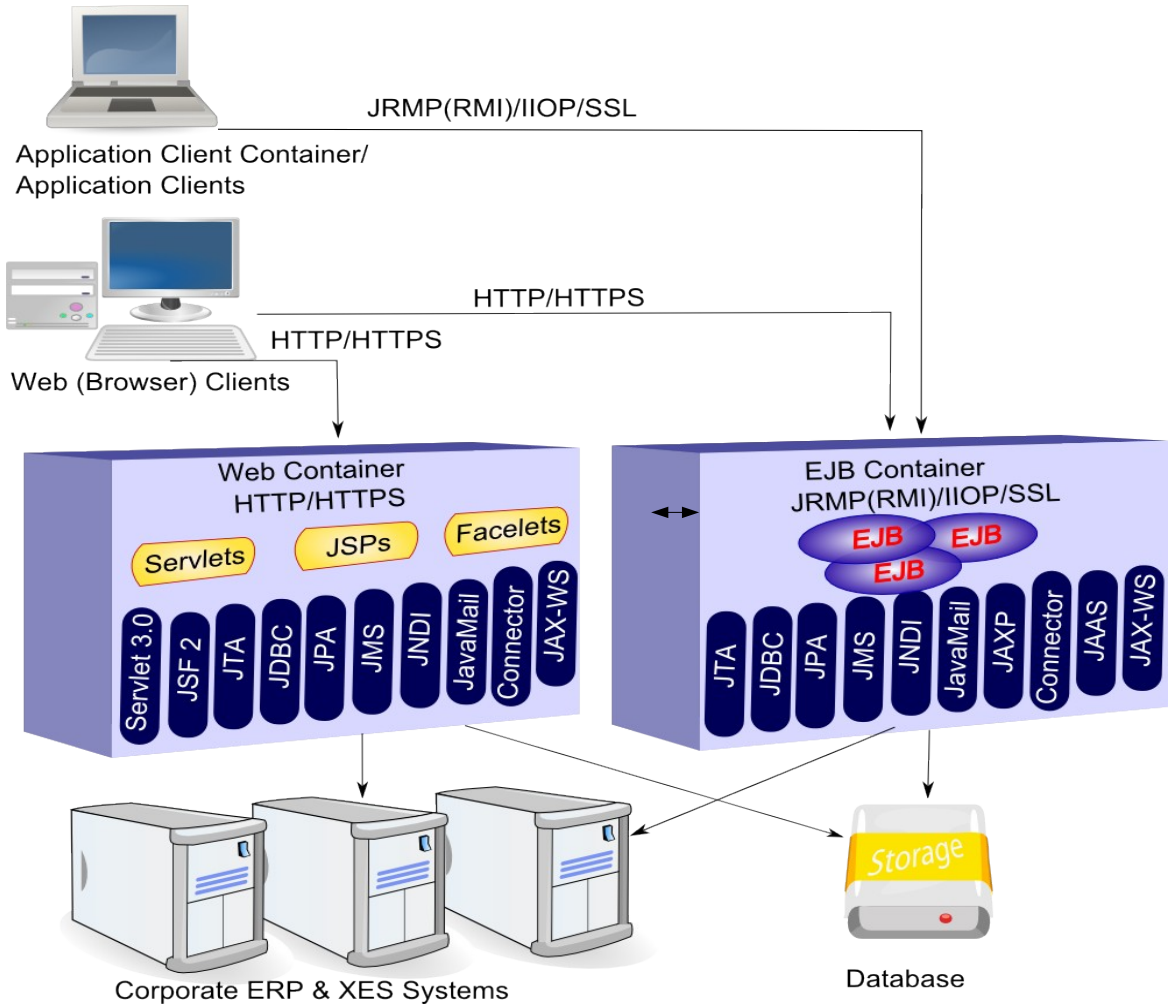
Views of the Architectural Model

- Functional/logic view
- Code/module view
- Development/structural view
- Concurrency/process/thread view
- Physical/deployment view
- User action/feedback view
- Data view

Architectural Styles

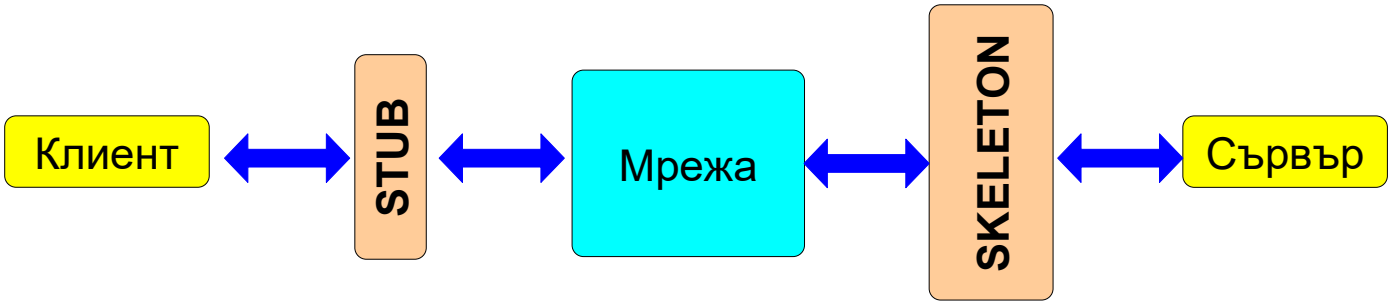
- Blackboard
- Client-server (2-tier, 3-tier, n-tier, cloud computing)
- Component-based
- Event-driven (or implicit invocation)
- Layered (or multilayered architecture)
- Microservices architecture
- Monolithic application
- Peer-to-peer (P2P)
- Pipes and filters
- Plug-ins
- Reactive architecture
- Service-oriented / Representational state transfer (REST)
- Rule-based

JavaEE Architecture



RMI Main Concepts

- RMI = Remote Method Invocation
- RMI – Remote Procedure Calls (RPC) implementation in pure Java



- Java client to Java server – simple in comparison with Common Object Requesting Broker Architecture (CORBA)

  Примерна IDL дефиниция:

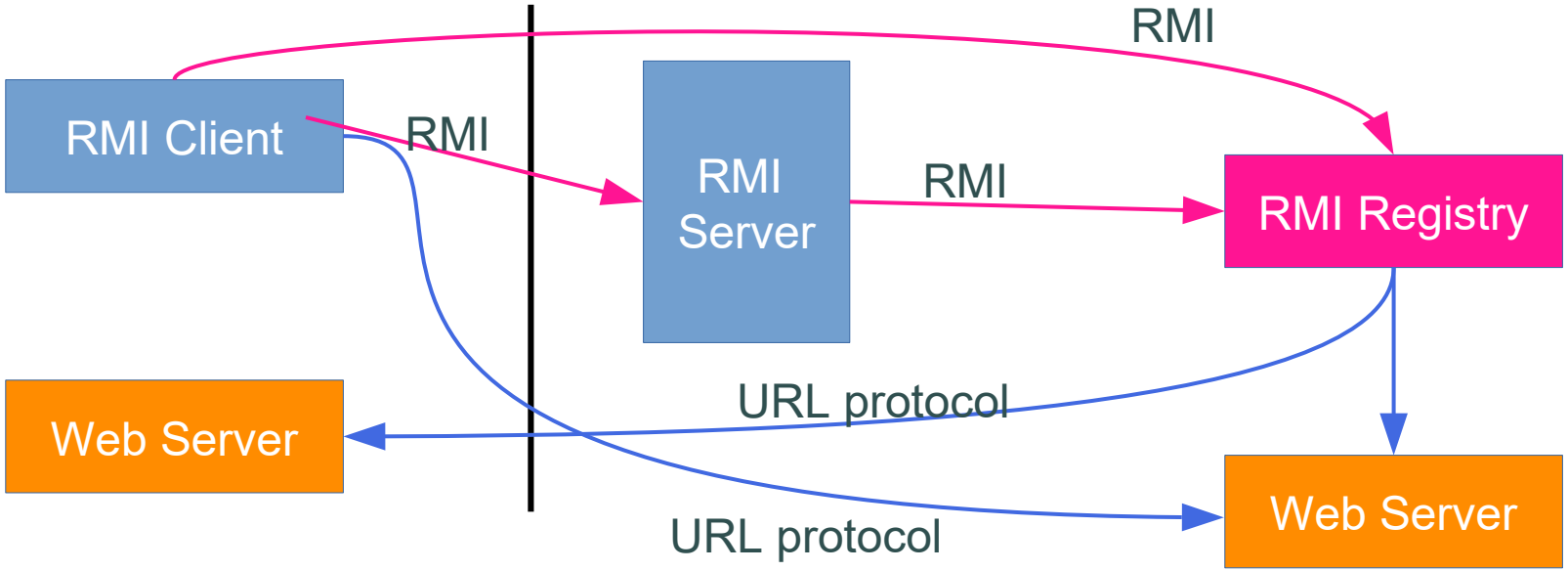
  interface getPrice {float calculate_price (in float amount); }

Remote Interfaces, Objects, and Methods

Remote objects – implement a remote interface, with following characteristics:

- extends the interface java.rmi.Remote;
- each method declares java.rmi.RemoteException in its throws clause, in addition to application-specific exceptions;
- clients invoke methods on a local stub (proxy), which is responsible for carrying out the method invocation on the remote object;
- a stub for a remote object implements the same set of remote interfaces that the remote object implements.

RMI Architecture



Distributed object applications need to do the following:

- Locate remote objects
- Communicate with remote objects
- Load class definitions for objects that are passed around

Dynamic Code Loading

- RMI can download remote objects' class definitions.
- Behaviors of an object, available in a single JVM, can be transmitted to another remote JVM.
- RMI loads actual classes of the objects, so their behaviors are not changed when they are sent to another JVM machine.
- Dynamic extension of RMI client's functionality by downloading new types/behaviors.
- The compute engine demo uses this capability to introduce new behaviors in a distributed computations system.

Main Implementation Steps

1. Define the remote interface declaring all business methods that will be called remotely from other virtual machines
2. Develop server class that implements the defined remote interface
3. Register the remote object in a RMI Registry
4. ~~Compile the Stubs and Skeletons using RMI compiler – rmic~~
5. Lookup of the remote object by the client using RMI Registry
6. Invocation of the remote methods by the client
7. Setting up a SequrityManager and defining security policies (by specifying appropriate options when starting the JVM).

Defining the Remote Interface

- The remote interface should be public
- Interface should extend java.rmi.Remote
- Interface methods should declare:

  throws java.rmi.RemoteException
- The remote object accessible from the RMI client side should be declared of the interface type, not the sever implementation type.

Implementing the Remote Interface and RMI Registry registration

- It is necessary the code that creates and registers the implementing class instance in the RMI Registry to be started using SequrityManager.
- The class implementing the Remote interface should be exported using java.rmi.server.UnicastRemoteObject
- The created new instance should be registered. If the RMI Registry service is not already started => it can be started as follows:
  - Manual: start rmiregistry 2009
  - Programmatic: LocateRegistry.createRegistry(2009)

Implementing the Remote Interface and RMI Registry registration

```
if (System.getSecurityManager() == null) {
   System.setSecurityManager(new SecurityManager());
}
try {
    Registry registry =
            LocateRegistry.createRegistry(REGISTRY_PORT);
     // Registry registry = LocateRegistry.getRegistry("127.0.0.1", REGISTRY_PORT);
     String name = "ComputationServer";
     Compute engine = new ComputeServer();
     Compute stub =
            (Compute) UnicastRemoteObject.exportObject(engine, 0);
     registry.rebind(name, stub);
     System.out.println("ComputeServer started");
} catch (Exception e) {  e.printStackTrace(); }
```

Implementing the Remote Interface and RMI Registry registration

Dynamically generated by constructing an instance of a Proxy with the following characteristics:

- The proxy's class is defined by the class loader of the remote object's class.
- The proxy implements all the remote interfaces implemented by the remote object's class.
- The proxy's invocation handler is a RemoteObjectInvocationHandler instance constructed with a RemoteRef.
- If the proxy could not be created, a StubNotFoundException will be thrown.

Compiling static stubs – Deprecated!

- RMI comiler – rmic – generationg static stubs (& skeletons) – Example:

```
rmic -classpath "C:\Java Course\Work\OnlineStore"
      -d "C:\Java Course\Work\OnlineStore" ProductController
```

- After RMI 1.2 skeletons are not created – only stubs
- Static stubs can be distributed with the client – deprecated!
- Класовете can be loaded dynamically (URL) from the RMI Registry – JVM option when running the client:

```
-Djava.rmi.server.codebase=
 file:/c:/home/trayan/public_html/classes/contract.jar
```

Remote Object Lookup and Use by the Client

At client side:

```
ProductController pc =
    (ProductController) Naming.lookup(
                "//localhost:1099/ProductController");
List<Product> lp = pc.getProducts();
```

or:

```
Registry registry =
        LocateRegistry.getRegistry("localhost:1099");
ProductController stub =
    (ProductController) registry.lookup("ProductController");
```

Dynamically Loading Stubs from Client Side

```
try {
    ProductController stub;
    ProductController pController = new ProductControllerImpl();
    System.setSecurityManager(new RMISecurityManager());
    Registry registry = LocateRegistry.createRegistry(1099);
    ProductController stub =
            (ProductController) UnicastRemoteObject
                    .exportObject(pController, 0);
    registry.rebind("//localhost:1099/ProductController", stub);
} catch (Exception e) {
    e.printStackTrace();
}
```

Security Policy Configuration

- Файл с политики за сигурност – виж C:\Program Files\Java\jdk1.8.0_181\jre\lib\security**java.policy**
- Задаване на позволения (Permissions) - пример:

```
grant codeBase "file:///C:/myapp/sysadmin/" {
    permission java.io.FilePermission "/resources/abc", "read";
};
```

- Инструмент за създаване и редактиране на файла с политики – policytool

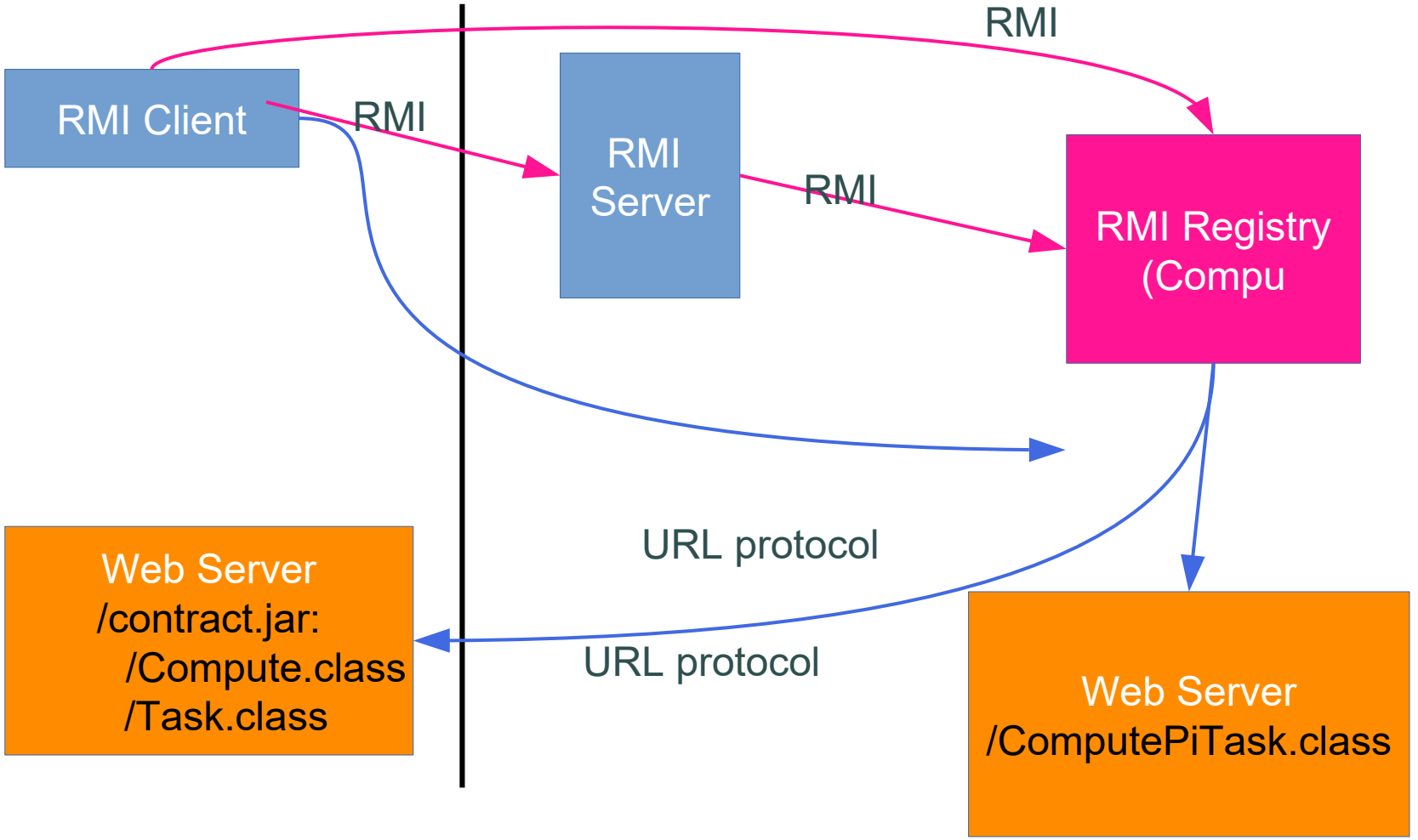
```
policy.all - grant {  permission java.security.AllPermission; };
```

- Опции на Java VM: -Djava.security.policy="policy.all"

Remote Exception Handling

```
if (System.getSecurityManager() == null) {
    System.setSecurityManager(new SecurityManager());
}
try {
    Registry registry = LocateRegistry.getRegistry(args[0],
    Integer.parseInt(args[1]));
    Compute comp = (Compute) registry.lookup("ComputationServer");
    ComputePiTask task = new
    ComputePiTask(Integer.parseInt(args[2]));
    BigDecimal pi = comp.executeTask(task);
    System.out.println(pi);
} catch (RemoteException e) {
    System.err.println("ComputePiTask exception:");
    e.printStackTrace();
} catch (NotBoundException e) {
    System.err.println("Server not registered with RMI Registry:");
    e.printStackTrace();
}
```

Building sample RMI client-server application: ComputeServer



References

- Software architecture in wikipedia: http://en.wikipedia.org/wiki/Software_architecture
- Java™ RMI Technology documentation page at Oracle® website: http://docs.oracle.com/javase/8/docs/technotes/guides/rmi/
- Oracle® Java™ RMI Simple Tutorial - https://docs.oracle.com/javase/tutorial/rmi/overview.html
- Security Features in Java SE - https://docs.oracle.com/javase/tutorial/security/index.html

Thank's for Your Attention!

Trayan Iliev

CEO of IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD

https://plus.google.com/+IproductOrg